[MVA] - Rapport de stage

Mathis Reymond

April 2024

Goodhart's law

When a metric becomes a target,
it ceases to be a good metric.

1 Introduction

Continual learning is a machine learning paradigm that focuses on the ability of a model to learn and adapt to new information over time, without forgetting previously learned knowledge. It is particularly relevant in scenarios where data is non-stationary or arrives in a sequential manner. By leveraging techniques such as online learning, regularization, and knowledge distillation, continual learning enables models to continuously update their knowledge and improve performance as new data becomes available.

1.1 Initial project

1.1.1 Project statement

In this context, the aim of the project was initialy stated as follows:

> *Catastrophic forgetting is a major challenge in continual learning systems, where a neural network needs to adapt to new tasks that differ from what it has previously learned. To tackle this, experts have devised strategies to either keep the neural network's critical weights intact or use generative models to refresh the network's memory of past tasks. However, these methods aren't perfect for hardware implementation due to their additional memory demands or the need for a separate network that constantly rehearses past information.*
>
> *Meanwhile, we understand that a system's architecture significantly impacts its ability to generalize knowledge. Enter Gradmax [4], an innovative algorithm that simultaneously optimizes a network's structure and weights, and smartly allocates neurons for new tasks without disrupting existing knowledge. This method is memory and computation efficient, making it suitable for hardware applications. But it does require hardware capable of dynamically adjusting its architecture to integrate new neurons as needed. Our recent work has led to a flexible hardware architecture, Mosaic [1], with multiple cores that can rearrange their connections in real-time.*
>
> *The aim of this project is to explore the integration of Gradmax with our dynamic 'Mosaic' architecture. We plan to approach this by factoring hardware limitations into the optimization process. In addition, we're considering the development of a gating network within this architecture that determines the optimal timing for assigning tasks to newly added neurons. This could leverage cutting-edge techniques like Mixture of Expert models [2].*

Let's dissect this project statement to understand the main objectives and challenges it encapsulates.

Computational costs. The project aims to address the computational costs associated with continual learning methods. Thus we expect to have a way to measure this cost, employ as figure of merit or intregrate it to the optimization process and to compare it to the cost of other methods.

Continual learning approach. Litterature categorizes continual learning approaches into three main categories: rehearsal, regularization, and architectural. However in the process of taking into account the computational costs, we will explore the architectural approach.

Gradmax Gradmax is a method designed for learning on a single task. Thus this project invites to extend it to continual learning.

Mosaic In the process of adapting Gradmax method to the continual learning framework, we will have to consider the hardware constraints of the Mosaic architecture.

1.1.2 Major issues

In the very beginning of the project, we faced two major issues. The first one is a consequence of Gradmax paper misleading introduction. Indeed, in the very first paragraph of the paper, the authors introduces :

Searching for the best architecture for a given task is an active research area with diverse approaches.[...] Most of these approaches are costly, as they require large search spaces or large architectures to start with. In this work we consider an alternative approach.

The genuine expectation regarding the work following such an introduction is that the authors will provide an inovative method that factorizes architecture search in the optimization process of learning a task while avoiding to train numerous high-size architectures, which is computationally expensive.

However, contrary to expectations, GradMax falls short at enabling neural architecture optimization. The method consists in growing a network from a root architecture to a predetermined target architecture, by adding neurons to it through training epochs. It focuses on the initialization of added neurons (the *how*), neglecting the crucial questions of *when* and *where* to add them, which are deferred for future research. In the paper, the decisions regarding *where* and *when* to add neurons are predetermined and take the form of a schedule established. Consequently, it's challenging to conclude that the GradMax framework effectively enables architecture optimization for growing artificial neural networks.

Additionaly, authors do not consider the computational cost of their method in the rest of the paper. This is a major issue as the project was initially designed to reduce the computational cost of continual learning methods.

1.1.3 Additional hurdles

As a general concept, we considere that continual learning referes to the ability of a model to learn and adapt to new information over time, without forgetting previously learned knowledge. However, this wide definition encompasses a variety of scenarios. First, it can be the case that there is no explicit task boundary, and the model must continuously adapt to new data without any indication of when a new task begins. This is known as incremental learning. Second, the model may be presented with a sequence of tasks, each with its own training data and objectives. In this case, the model must learn each task sequentially and adapt to new tasks without forgetting the previous ones, which is known as sequential learning. Third, the model may be presented with a series of tasks that are related or have some shared structure. In this case, the model must leverage this shared structure to learn new tasks more efficiently. This is known as multitask learning.

But even after this specification, ambiguities persist. For instance when it comes to sequential learning, some frameworks assume that the model is knows when a new task arrives, whereas others do not make this assumption.

1.2 Revised project

2 Regularization approaches

2.1 LoRA

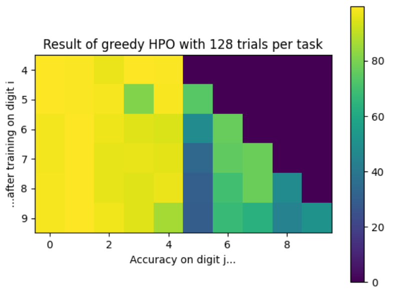
LoRA stands for Low Rank Adaptation. It is a method that aims to reduce the number of parameters that are updated when learning a new task. The idea is to factorize the weight matrix of a neural network into two low-rank matrices, which are shared across tasks, and a task-specific matrix. This allows the model to adapt to new tasks by updating only the task-specific matrix, while keeping the shared matrices fixed. This approach is motivated by the observation that neural networks tend to learn task-specific information in the last layers, while the lower layers capture more general features that are shared across tasks.

LoRA and refinement have been intensively used for finetunning. However, it would be convenient to be able to leverage this approach in the context of continual learning. Unfortunately, we cannot directly apply the fine-tunning approach to continual learning, as even if we have specialized sets of weight for each task, we would not know which set to used when given a new sample. Instead, we can realie on a core neural networks and update it each time we face a new task by adding LoRA weights trained specificly this new task.

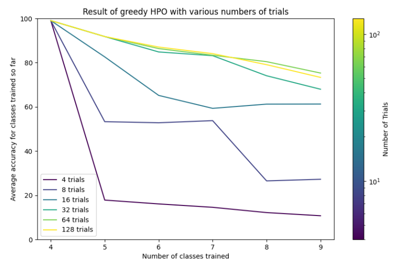
Algorithm 1 LoRA for continual learning

```
1: **Input:** A list of $n$ tasks $L = [(D_1, L_1), (D_2, L_2), \ldots, (D_n, Ln)]$
2: **Output:** A model $\phi$ able to solve each task
3:
4: **Step 1:** Initialize the core model $\phi$ and train it on $(D_1, L_1)$
5: **for** each task $(D_i, L_i)$ in the list **do**
6:     **Step 2:** Freeze parameters of $\phi$ and initialize LoRA parameters $\mathbb{L}_i$.
7:     **Step 3:** Train LoRA parameters on task $(D_i, L_i)$ using $\phi$ for forward propagation.
8:     **Step 4:** Update $\phi$ by adding $\mathbb{L}_i$ parameters.
9: **end for**
10: **Step 5:** The core model $\phi$ is now trained on each task
```

LoRA introduces a parameter α to set how important the LoRA weights are compared to the core weights.



(a) Evolution of test accuracy of each task through continual learning



(b) Average test accuracy for tasks trained so far, at various HPOs expenses

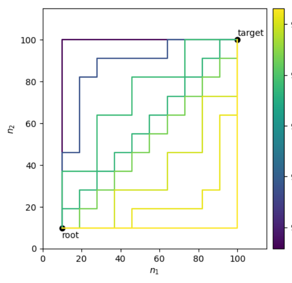
Figure 1: LoRA resuls

We demonstrate that the success of this approach heavily relie on the hyperoptimization performed.

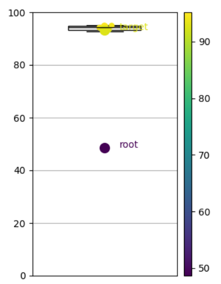
2.2 Growth in continual learning

2.2.1 Motivating growth exploration

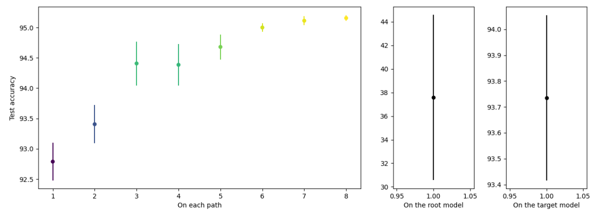
Trying to taclke the questions of where and when to grow neurons in the framework of GradMax, that is, starting with a root architecture and converging to a target architecture, it can be interesting to assess whether or not the way we grow the architecture toward this target architecture matters. For example, we can considere a simple two-hidden layer architecture, and grow it by adding neurons to the first hidden layer first, then to the second hidden layer, or the other way around. So the order we add neurons to each layer define a path. And we can ask ourselves if some paths are more efficient than others.



(a) Average test accuracy achieved by following each paths to grow the network, accross 5 trials



(b) Average test accuracies of the paths, the root model and the taregt model, accross 5 trials



(c) Average test accuracy across 5 trials, with standard deviation

Figure 2: Results on MNIST with GradMax initialization

To do so, we set the root network (RN) to be a two-hidden-layer MLP with 10 neurons on each layer, and the target network (TN) to be a two-hidden-layer MLP with 100 neurons on each layer. As reference for comparison, we train both RN and TN on 300 batch MNIST batches, each of which is of size 128, for 3 epochs. We use Adam optimizer with learning rate equal to 5×10^{-3}, and mean squarred error loss function. We repeat this process 5 times for statistical reliability. See 2c for the results.

Then, in the processus of growing RN toward TN through the training process, we define schedules. Each schelule consists 20 growth step. 10 of these growth steps consist in adding 9 neurons to the first and the other 10 growth steps consist in adding 9 neurons the second hidden layer. Thus, the only way two schedules differ from each other is this order it plans to grow each layer. As for training RN and TN, the growing process occurs while training on 300 batch MNIST batches, each of which is of size 128, for 3 epochs. Overall, each growth of 10 neurons happens once every We use Adam optimizer with learning rate equal to 5×10^{-3}, and mean squarred error loss function.

That being said, we can see that the path we choose to grow the network has a significant impact on the final test accuracy. In particular, the path that consists in adding neurons to the first hidden layer first, then to the second hidden layer, seems to be the most efficient.

Thus, exploring neurogeneis makes sense in the context of traditional learning (on a single task). However, what we want to do is to adapt this to a contiual learning framework. That is, we want to grow the network

2.3 EWC

Issues with the fundational paper itself :

- No code avaiable
- No specification of the number of λ parameters
- No specification of the values of the λ parameters
- Mismatch between figure and references to figures

Issues with the approach :

- Ranks on paperwithcode benchmarks : $\frac{5}{6}, \frac{13}{15}, \frac{6}{8}, \frac{6}{6}, \frac{3}{7}$, so the approach falls just above the first quintile, ranking on average 22th out of 100 approaches. But when we read the paper, it feels like continual learning is solved.
- Redundancy
- 2 to 1 pollution
- 1 to 2 pollution
- Too much hyper parameters (lambda(s))
- Fact that no matter how hard you try, every time you learn a new task, you do hurt knowledge about the previous tasks. So from the very beginning, the approach is doomed to be very limited... the only way to save the face is to overfit with loads of HPs.

2.4 Conclusions we can draw from these attempts

2.4.1 Limitation of approaches

As a very general principle, the problem with these approaches in the context of continual learning is that

Improving on task 2 mean regressing on task 1, and reciprocally

Which implies that performing continual learning in such a way requires to perform trade-offs. And the most convenient way one performs trade-offs in deep learning is through hyperparameters.

We often assume that it is fine to attempt somewhat limited approaches because improvements are achieved by building on failed attempts. This is true for incremental improvements, typically made by engineers through the combination and optimization of previous work. However, researchers aim for disruptive improvements by stepping back from the current state of the art to identify what is arbitrary, poorly motivated, or inherently limited in current approaches.

Let's assume we are given I tasks indexed by the set $[\![1, I]\!]$. We define a neural network as the set of its parameters

$$\phi = \{\{w_{l,m,n} \in \mathbb{R} | m, n \in [\![1, N_{l-1}]\!] \times [\![1, N_l]\!]\} \, | l \in [\![1, L+1]\!]\} \tag{1}$$

where L is the number of hidden layers and N_l is the number of neurons on layer l.
Let's define $\mathbb{W}^i_{l,n}$ to be the set of protected weights for task i with respect to neuron n of layer l, for any $i \in [\![1, I]\!]$, $l \in [\![1, L+1]\!]$ and $n \in [\![1, N_l]\!]$:

$$\mathbb{W}^i_{l,n} = \{w_{l,m,n}, m \in [\![1, N_{l-1}]\!] | w_{l,m,n} \text{ protected}\} \tag{2}$$

which allows to define the pollution of task i by protected weights for task j on neuron n of layer l as

$$p^{i \to j}_{l,n} = \mathbb{E}_{x \sim D_i} \left[\frac{\sum_{w \in \mathbb{W}^j_{l,n}} w(x)}{\sum_{w \in \mathbb{W}^i_{l,n}} w(x)} \right] \tag{3}$$

Finally, we can define the pollution of task i by protected weights for task j on the whole network as

$$P^{i \to j} = \frac{1}{L+1} \sum_{l=1}^{L+1} \frac{1}{N_l} \sum_{n=1}^{N_l} p_{l,n}^{i \to j} \tag{4}$$

To this regard, as a general concept, we can control :

- how the added neurons react to new task samples
- how the old neurons react to new task samples
- (depending on the approach) how the old neurons react to previous tasks samples

However, we have absolutely no control over the way the added neurons react to previous tasks samples because :

- we have no acces to previous tasks samples
- we are not using any rehearsal strategy
- we do not intend to encode any information about the specificity of previous tasks in the weights

So our very last hopes without memory are :

- to parametrize/constrain the weights so that they encode task information

Interestingly, these questions and remarks we have been objecting to ourselves about fundations of our approach applies to several other approaches including :

- regularization approaches such as EWC, Hessian and Bayesian approaches
- our LoRA approach
- neural growth approaches (whether initialization is done randomly, through GradMax or TINY)
- use of distincts neural networks for each task

However, when reading papers about these approaches we have the general feeling that continual learning is a solved problem whereas we argued that they are not adressing continual learning. While they have no information about previous tasks, these approaches can only positive results by performing trade offs between the total number of tasks in the benchmark, how much you forget about the previous tasks and how much you can learn about the new one at hand. That is, these approaches heavily rely on hyperparameter optimization.

So in a nutshell, these approaches succeed at solving one specific continual learning benchmark at a time. None of these approaches can perform well on a new benchmark without going through a new heavy hyperparameter optimization process. So, can we really considere them as general continual learning approaches ? Because there seem to a close ressemblance to overfitting ; a hyper-overfitting.

Would we need to need to define a validation set of benchmarks ? And require every approaches to be able to run on test benchmarks with none of few hyperparameter updates ?

2.4.2 Lack of specificity and definitions

In addition to the previous

no definition of : - overfitting on a benchmark - define and measure the hyper parameter importance (or cost) of any given approach, to quantify overfitting - task - similarity between tasks

References

[1] Thomas Dalgaty et al. “Mosaic: in-memory computing and routing for small-world spike-based neuromorphic systems”. In: *Nature Communications* 15.1 (Jan. 2024). ISSN: 2041-1723. DOI: `10.1038/s41467-023-44365-x`. URL: `http://dx.doi.org/10.1038/s41467-023-44365-x`.

[2] David Eigen, Marc’Aurelio Ranzato, and Ilya Sutskever. *Learning Factored Representations in a Deep Mixture of Experts*. 2014. arXiv: `1312.4314 [cs.LG]`.

[3] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: `2109.12894 [cs.NE]`.

[4] Utku Evci et al. *GradMax: Growing Neural Networks using Gradient Information*. 2022. arXiv: `2201.05125 [cs.LG]`.

[5] Ziming Liu et al. *Growing Brains: Co-emergence of Anatomical and Functional Modularity in Recurrent Neural Networks*. 2023. arXiv: `2310.07711 [q-bio.NC]`.

[6] Javier Lopez Randulfe and Leon Bonde Larsen. *A multi-agent model for growing spiking neural networks*. 2020. arXiv: `2010.15045 [cs.NE]`.

[7] Xin Yuan, Pedro Savarese, and Michael Maire. *Accelerated Training via Incrementally Growing Neural Networks using Variance Transfer and Learning Rate Adaptation*. 2023. arXiv: `2306.12700 [cs.LG]`.

Appendix

2.5 Other results for motivating exploration of growth paths

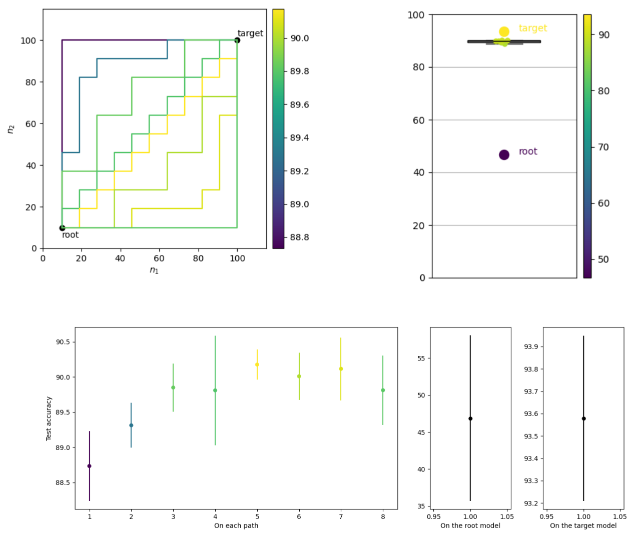


Figure 3: Results on MNIST with random initialization

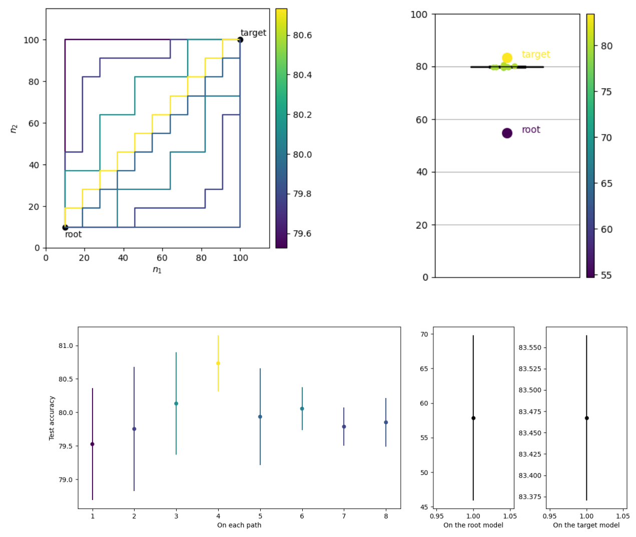


Figure 4: Results on FMNIST with random initialization

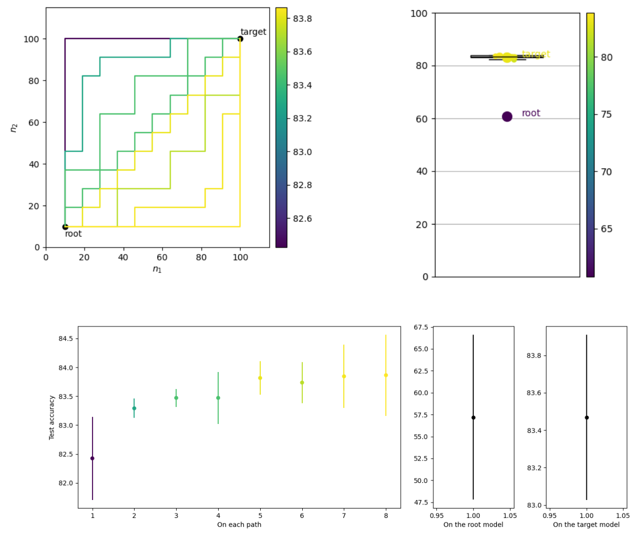


Figure 5: Results on FMNIST with gradmax initialization

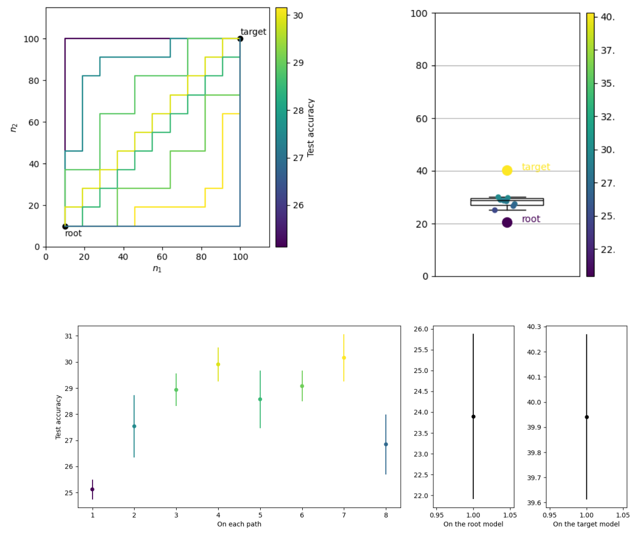


Figure 6: Results on CIFAR10 with random initialization

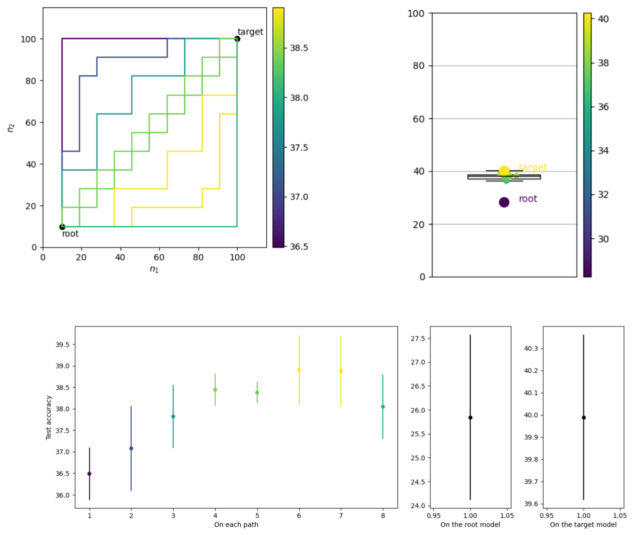


Figure 7: Results on CIFAR10 with gradmax initialization